Понятие «Чистый код».

Содержательные имена.

Подготовил Соловьев М.Д.

Чистый код.

Конкретного определения «Чистого кода» не существует. Не во всех средах программирования есть всеми признанный («единственно верный») кодекс аккуратности, иногда его просто нет или существует несколько конкурирующих. Поэтому, зачастую «чисто» написанный одним программистом код, покажется другому грязным.

Признаки.

Существуют признаки «чистого кода»:

* Код легко читается и понимается.

* Код легко поддаётся изменениям.

* Код может быть расширен, либо встроен куда-нибудь в виде отдельного модуля.

* Код поддаётся автоматизированному тестированию.

Как написать красивый и чистый программный код?

Для того, чтобы код выглядел чистым (что по сути обеспечивает нормальную работу в команде), нужно следовать нескольким простым правилам, которыми никогда не нужно пренебрегать.

Тщательно продумайте логику, перед тем как начать кодить.

Перед тем как начинать процесс написания кода или его отладки, возможно целесообразно написать какой-то псевдо код для того, чтобы сразу представить себе весь масштаб того, что вам предстоит написать. Если вы будете следовать данному правилу, то нет сомнений в том, что читабельность и функциональная сторона вашего кода значительно улучшится. Кроме всего прочего данная техника позволит вам избежать ненужных дополнений, которые как правило впоследствии добавляются в код.

Используйте понятные идентификаторы.

Для того чтобы пример рассмотренной вами структуры был понятен почти каждому после нескольких секунд, необходимо давать ясные имена идентификаторов. Также необходимо принимать во внимание форматирование, которое также играет важную роль в читабельности кода. Согласитесь, что это не очень сложно! Однако данные техники помогут вам, и не раз.

Использование «хороших» имен – это самая важная привычка, поскольку содержательные имена упрощают чтение и понимание программного кода. Понятность вашего кода в конечном итоге определяет возможность его последующего сопровождения. Даже если написанный вами код не будет содержать комментариев, но при этом будет прост для понимания, то вам или кому-либо еще будет легче его изменить в случае необходимости. При выработке этой привычки ваша цель должна состоять в том, чтобы благодаря «хорошим» именам ваш код читался так же легко, как книга.

Неоднозначные или бессмысленные имена.

```php
<?php

function getNBDay($d)
{
    switch($d) {
        case 5:
        case 6:
        case 7:
            return 1;
        default:
            return ($d + 1);
    }
}

$day = 5;

$nextDay = getNBDay($day);

echo ("Next day is: " . $nextDay . "\n");

?>
```

Пример кода, содержащий слишком короткие имена переменных, сложные для понимания сокращения и имена методов, которые не описывают производимые ими действия. Имя метода, намекающее на какие-либо выполняемые этим методом действия, в то время как в действительности он делает что-то совершенно другое, может оказаться чрезвычайно вредным, поскольку будет вводить в заблуждение.

Осмысленные и лаконичные имена.

```php
<?php

define ('MONDAY', 1);
define ('TUESDAY', 2);
define ('WEDNESDAY', 3);
define ('THURSDAY', 4);
define ('FRIDAY', 5);
define ('SATURDAY', 6);
define ('SUNDAY', 7);

/*
 *
 * @param $dayOfWeek
 * @return int Day of week, with 1 being Monday and so on.
 */
function findNextBusinessDay($dayOfWeek)
{
    $nextBusinessDay = $dayOfWeek;

    switch($dayOfWeek) {
        case FRIDAY:
        case SATURDAY:
        case SUNDAY:
            $nextBusinessDay = MONDAY;
            break;
        default:
            $nextBusinessDay += 1;
            break;
    }

    return $nextBusinessDay;
}

$day = FRIDAY;

$nextBusDay = findNextBusinessDay($day);

echo ("Next day is:" . $nextBusDay . "\n");

?>
```

Данный код демонстрирует использование хороших привычек именования. Измененные имена методов лучше отражают, что делают эти методы и почему. Имена переменных также стали более содержательными. Единственная переменная, которая осталась короткой в этом листинге – это переменная цикла $i.

Пишите краткие и понятные комментарии.

Комментарии - это очень важная часть программной документации, которую необходимо писать, для того чтобы быстро и безболезненно ориентироваться в коде. Простой и краткий комментарий это как раз то, что вам нужно, для того чтобы увеличить понимание того, что следует за ним.

Многие разработчики не любят писать комментарии. Это вполне можно понять, однако часто это просто необходимо, потому что всё чаще и чаще масштабы и количество проектов увеличивается и всё поместиться в голове не может. Но тут есть и обратная сторона медали, но об этом в следующем пункте.

Используйте комментарии без фанатизма.

Использование комментариев не должно выходить за рамки. Когда вы пишите комментарий, то должны просто описать входящие и возвращаемые данные. Комментарии следующего вида неприемлемы:

- Написание комментариев для себя (пример: /* Закончу как-нибудь потом… */).
- Отвод глаз от себя (пример: /* это писал Johns.C него и спрос. */).
- Ни о чём не говорящие выражения (e.g. /* Это очередная математическая функция. */).
- Также иногда люди не уверены в какой-то функциональности и просто комментируют фрагмент кода.

Данные комментарии бессмыслены и могут ввести человека в заблуждение. У каждого комментария должен быть свой смысл. Никогда не забывайте об этом

Примеры хороших комментариев:

- Спецификация автора (пример: /* Автор John, 13 Ноября 2010 */).
- Какая-то детализация метода или процедуры (пример: /* Данный метод предназначен для валидации формы входа пользователя в систему */).
- Быстрые заметки или описание изменений в коде (e.g. /* Добавлена валидация e-mail */).

Отсутствующее и избыточное документирование функций.

```php
<?php

class ResultMessage
{
    private $severity;
    private $message;

    public function __construct($sev, $msg)
    {
        $this->severity = $sev;
        $this->message = $msg;
    }

    public function getSeverity()
    {
        return $this->severity;
    }

    public function setSeverity($severity)
    {
        $this->severity = $severity;
    }

    public function getMessage()
    {
        return $this->message;
    }

    public function setMessage($msg)
    {
        $this->message = $msg;
    }
}

function cntMsgs($messages)
{
    $n = 0;
    /* iterate through the messages... */
    foreach($messages as $m) {
        if ($m->getSeverity() == 'Error') {
            $n++; // add one to the result;
        }
    }
    return $n;
```

Комментарии в примере просто описывают, что делает код – осуществляет итерацию цикла или добавление числа. Однако при этом отсутствует
объяснение, *почему* выполняется именно это. Тому, кто в будущем станет сопровождать этот код, будет непросто понять, сможет ли он безопасно изменить этот код.

Документирование функций и классов.

```php
<?php
/**
 * The ResultMessage class holds a message that can be returned
 * as a result of a process. The message has a severity and
 * message.
 *
 * @author nagood
 *
 */
class ResultMessage
{
    private $severity;
    private $message;

    /**
     * Constructor for the ResultMessage that allows you to assign
     * severity and message.
     * @param $sev See {@link getSeverity()}
     * @param $msg
     * @return unknown_type
     */
    public function __construct($sev, $msg)
    {
        $this->severity = $sev;
        $this->message = $msg;
    }

    /**
     * Returns the severity of the message. Should be one
     * "Information", "Warning", or "Error".
     * @return string Message severity
     */
    public function getSeverity()
    {
        return $this->severity;
    }

    /**
     * Sets the severity of the message
     * @param $severity
     * @return void
     */
```

Комментарии в примере говорят читателю о назначении классов и методов. Они говорят, зачем функции делают то, что они делают. Это будет весьма полезно в будущем, в процессе сопровождения этого кода. Изменившиеся условия могут потребовать модификации кода. Произвести изменения будет гораздо легче, если назначение кода будет просто найти.

Не пишите слишком большие методы.

Когда функциональность того или иного метода расширяется, неизбежно растёт и размер самого метода. Количество строк кода постоянно увеличивается, и метод меняет свой вид, что может в последствии запутать автора.

Одним из решений данной задачи является разбиение одной большой функции на несколько более мелких. Ведь часто код в некоторых функциях повторяется, и его можно заменить другой функцией – просто нужно не лениться. Во всём этом есть и другие плюсы, скажем, другой разработчик в команде сможет использовать ваши методы для своих. Всё это нужно делать очень осторожно.

Длинные функции.

```php
<?php

function writeRssFeed($user)
{
    // Get the DB connection information


    // look up the user's preferences...
    $link = mysql_connect('mysql_host', 'mysql_user', 'mysql_password')
        OR die(mysql_error());

    // Query
    $perfsQuery = sprintf("SELECT max_stories FROM user_perfs WHERE user= '%s''
            mysql_real_escape_string($user));

    $result = mysql_query($query, $link);

    $max_stories = 25; // default it to 25;

    if ($row = mysql_fetch_assoc($result)) {
        $max_stories = $row['max_stories'];
    }

    // go get my data
    $perfsQuery = sprintf("SELECT * FROM stories WHERE post_date = '%s'",
            mysql_real_escape_string());

    $result = mysql_query($query, $link);


    $feed = "<rss version=\"2.0\">" .
        "<channel>" .
        "<title>My Great Feed</title>" .
        "<link>http://www.example.com/feed.xml</link>" .
        "<description>The best feed in the world</description>" .
        "<language>en-us</language>" .
        "<pubDate>Tue, 20 Oct 2008 10:00:00 GMT</pubDate>" .
        "<lastBuildDate>Tue, 20 Oct 2008 10:00:00 GMT</lastBuildDate>" .
        "<docs>http://www.example.com/rss</docs>" .
        "<generator>MyFeed Generator</generator>" .
        "<managingEditor>editor@example.com</managingEditor>" .
        "<webMaster>webmaster@example.com</webMaster>" .
        "<ttl>5</ttl>";

        // build the feed...
```

Пример длинной функции. Такая функция является источником потенциальных проблем по нескольким причинам. Она выполняет много несвязанных действий. Ее трудно понять, отладить и протестировать. Она совершает итерации и строит список элементов, она присваивает значения объектам, она выполняет вычисления и т.д.

Управляемые, конкретные функции.

```php
<?php

function createRssHeader()
{
    return "<rss version=\"2.0\">" .
        "<channel>" .
        "<title>My Great Feed</title>" .
        "<link>http://www.example.com/feed.xml</link>" .
        "<description>The best feed in the world</description>" .
        "<language>en-us</language>" .
        "<pubDate>Tue, 20 Oct 2008 10:00:00 GMT</pubDate>" .
        "<lastBuildDate>Tue, 20 Oct 2008 10:00:00 GMT</lastBuildDate>" .
        "<docs>http://www.example.com/rss</docs>" .
        "<generator>MyFeed Generator</generator>" .
        "<managingEditor>editor@example.com</managingEditor>" .
        "<webMaster>webmaster@example.com</webMaster>" .
        "<ttl>5</ttl>";
}

function createRssFooter()
{
    return "</channel></rss>";
}

function createRssItem($title, $link, $desc, $date, $guid)
{
    $item .= "<item>";
    $item .= "<title>" . $title . "</title>";
    $item .= "<link>" . $link . "</link>";
    $item .= "<description> " . $description . "</description>";
    $item .= "<pubDate>" . $date . "</pubDate>";
    $item .= "<guid>" . $guid . "</guid>";
    $item .= "</item>";
    return $item;
}

function getUserMaxStories($db_link, $default)
{
    $perfsQuery = sprintf("SELECT max_stories FROM user_perfs WHERE user= '%s'",
            mysql_real_escape_string($user));

    $result = mysql_query($perfsQuery, $db_link);
```

Более компактный, более удобочитаемый вариант предыдущего метода. В этом примере длинный метод разбит на методы меньшего размера, каждый из которых выполняет только одну задачу и делает это хорошо. Полученный в результате код проще повторно использовать в будущем, и проще тестировать.

Разделение длинных методов на более короткие имеет смысл до определенного предела, поэтому соблюдайте осторожность и не переусердствуйте. Вы можете разбить код так, что он останется таким же трудночитаемым, как и раньше, когда он был единой функцией.

Используйте стандарты именования переменных и функций.

Когда вы начинаете писать функцию или создаёте новую переменную, её имя должно иметь описательный характер. Разработчик с первого взгляда должен понять, для чего она нужна.

Существует немало компаний, в которых узаконены свои собственные стандарты именования методов и переменных (пример: префикс 'int_' для всех переменных данного типа), но также наравне с ними существует масса компаний где вообще нет никаких стандартов. Из-за своей лени работники данных компаний часто делают одну и ту же работу вдвое дольше.

Осторожно производите все изменения.

Для того чтобы не нарушить какой-то другой метод, следует очень осторожно производить все изменения в коде и записывать их в соответствующий комментарий. Обо всём этом нужно говорить, потому что почти никто так не делает. Если вам необходимо добавить или удалить какой-то функционал, постарайтесь сделать это так, чтобы код остался понятным и чистым.

Так же необходимо учитывать следующее:

Сохраняйте код читабельным (пример: если вы добавляете условный оператор IF, то обратите внимание на форматирование кода).

Создавайте дополнительные комментарии.

Уважайте стандарты, согласно которым был написан данный метод.

Избегайте смесей различных языков программирования.

Например, CSS встроенные в теги и небольшие фрагменты JavaScript в середине кода - это неверные примеры смешивания языков программирования, которые часто встречаются в реальной жизни. Это на самом деле может привести к потери времени в будущем, когда необходимо будет произвести какие-то изменения.


СПАСИБО ЗА ВНИМАНИЕ
risovach.ru